EXHIBIT A

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND-ODESSA DIVISION

MALIKIE INNOVATIONS LTD., KEY PATENT INNOVATIONS LTD. Plaintiffs, v. MARA HOLDINGS, INC. (F/K/A MARATHON DIGITAL HOLDINGS, INC Defendant.	CASE NO. 7:25-cv-00222-DC-DTG JURY TRIAL DEMANDED

EXPERT DECLARATION OF DR. CETIN KAYA KOC IN SUPPORT OF MARA'S OPENING CLAIM CONSTRUCTION BRIEF

TABLE OF CONTENTS

I. INTRODUCTION

1. I have been retained by counsel for Defendant MARA Holdings, Inc. (f/k/a Marathon Digital Holdings, Inc.) ("MARA") as an expert witness in the above-captioned proceeding. The following declaration is based on my personal knowledge, and, if called as a witness, I could and would testify completely hereto.

2. I understand that Malikie Innovations Ltd. and Key Patent Innovations Ltd. ("Plaintiffs") have asserted numerous patents ("Asserted Patents") and claims ("Asserted Claims") against MARA, including:

a. U.S. Patent No. 8,788,827 ("'827 Patent"): Asserted claims 1-5;

b. U.S. Patent No. 10,284,370 (the "'370 Patent"): Asserted claims 1-5;

c. U.S. Patent No. 8,666,062 (the "'062 Patent"): Asserted claims 1-4, 6-7;

d. U.S. Patent No. 7,372,960 (the "'960 patent"): Asserted claims 3, 6;

e. U.S. Patent No. 7,372,961 (the "'961 patent"): Asserted claims, 1-7;

f. U.S. Patent No. 8,532,286 (the "'286 Patent"): Asserted claims, 1, 5-6, 9.

3. For this declaration, I have been asked to analyze and opine on technical issues relating to claim construction of disputed terms. I have reviewed the Asserted Patents and Asserted Claims from the perspective of a person of ordinary skill in the art. This declaration sets forth my analyses and opinions based on my knowledge and experience and the materials I considered.

4. I am being compensated on an hourly basis for my work on this case at my standard consulting rate of $800 per hour. I have received no additional compensation for my work on this case, and my compensation does not depend upon the contents of this declaration or the outcome of these proceedings.

A. Qualifications

5. I received a Bachelor's of Science and a Master's of Science in Electrical Engineering from Istanbul Technical University in 1980 and 1982, respectively. I received a Master's of Science in Electrical & Computer Engineering from University of California, Santa Barbara ("UCSB") in 1985. I received my Ph.D. in Electrical & Computer Engineering from UCSB in 1988.

6. I am currently a retired research professor in the Department of Computer Science and the College of Creative Studies at UCSB; my research activities however continue. I am also the founder and director of a Research Lab at UCSB under my name. Before joining UCSB, I was a professor at Oregon State University (1992–2007) and an assistant professor at the University of Houston (1988–1992). At Oregon State University, I established the Information Security Laboratory and received the Research Award for Outstanding and Sustained Research Leadership in 2001.

7. My current research focuses on cryptographic hardware and embedded systems, elliptic curve cryptography and finite fields, and deterministic, hybrid and true random number generators. My research is funded privately and also by the National Science Foundation ("NSF"), on aspects of cryptographic engineering, performed within Koç Lab (http://koclab.cs.ucsb.edu) composed of postdoctoral researchers, Ph.D. candidates, MS candidates, and undergraduate students.

8. I also co-founded in 2003, and am currently employed by, Cryptocode Inc. Cryptocode develops state of the art cryptographic hardware and software solutions, and also offers analysis, design, and training services in cryptographic engineering.

9. I further co-founded in 1999 the Workshop on Cryptographic Hardware and Embedded Systems ("CHES Workshop"). The CHES Workshop is the second largest

cryptography conference and the premier forum for presenting scientific advances in all aspects of cryptographic hardware and security of embedded systems. More than 400 engineers and scientists from over 30 countries participate in CHES, submitting nearly 150 papers every year, with an acceptance rate of less than 20%. I was the program co-chair and proceedings editor of the CHES Workshop from 1999-2003. I have been a permanent member of the CHES Steering Committee, in addition to serving as the Publicity Chair, the General Chair, and the Program Committee member since its founding in 1999. In 2010, 2013, and 2016, Crypto and CHES conferences were held together at University of California Santa Barbara, and I served as the General Chair.

10. I am also the co-founder of two additional cryptography-related conferences: International Workshop on the Arithmetic of Finite Fields (WAIFI) and Security Proofs for Embedded Systems (PROOFS). WAIFI is a forum of engineers and mathematicians interested in efficient software and hardware realizations of finite fields. PROOFS is intended to promote methodologies that increase the confidence level in the security of embedded systems, especially those that contain cryptographic mechanisms. I am on the steering committee of both WAIFI and PROOFS, and was the program co-chair of WAIFI 2008 and the general co-chair of WAIFI 2010. I was the general chair of PROOFS in 2013, which took place at UCSB following the CHES Workshop.

11. I organized and chaired the Open Problems in Mathematical and Computational Sciences Conference, held in Istanbul on September 18-20, 2013. I also organized and chaired a workshop Cyber-Physical Security Education, in July 17-19, 2017 in Paris, France.

12. I am also the founding Editor-in-Chief of the Journal of Cryptographic Engineering (JCEN), which covers all aspects of design and implementation of cryptographic hardware and

software, including the research areas of the CHES Workshop. The Journal is published by the Springer publishing company, which also publishes all CRYPTO, EUROCRYPT, CHES Conference proceedings, and hundreds of books covering topics in information security and cryptography every year. The Journal is the only comprehensive source for scientific articles on methods, techniques, tools, implementations, and applications of research in cryptographic engineering, and presents articles discussing innovations in cryptographic hardware, cryptographic embedded systems, and embedded security.

13. I have also been on the editorial boards of IEEE Transactions on Computers (2003-2008 and 2015-2019) and IEEE Transactions on Mobile Computing (2003-2007). I was a guest co-editor of the April 2003, November 2008, and November 2018 issues of the IEEE Transactions on Computers on post-quantum cryptography, cryptographic and cryptanalytic hardware and embedded systems. In 2007, I was elected as IEEE Fellow for my contributions to cryptographic engineering. Furthermore, I was an Associate Editor of the prestigious International Journal of Foundations of Computer Science between 2016 and 2021.

14. I also co-authored five books, "Cryptographic Algorithms on Reconfigurable Hardware," "Cryptographic Engineering," "Open Problems in Mathematics and Computational Science," "Cyber-Physical Systems Security", and "Partially Homomorphic Encryption," published by Springer in 2007, 2009, 2014, 2018, and 2021, respectively. In addition to contributing to seven (7) conference proceedings as co-editor, I have also authored or co-authored more than 200 scientific papers, and am an inventor or co-inventor of 13 U.S. patents. I graduated twenty-one (21) Ph.D. students and forty (40) M.S. students, and also directed research theses of six (6) undergraduate students. Eleven (11) of my Ph.D. students are currently professors: 3 in the

U.S., 1 in Mexico, and 7 in other countries, and the remaining work for global high-tech companies in the U.S.

15. A copy of my curriculum vitae, which describes in further detail my qualifications, responsibilities, employment history, honors, awards, professional associations, and publications is submitted with this declaration, attached as Appendix A.

B. Materials Considered

16. Below are the materials I have considered in this declaration.

Exhibit	**Description**
Exhibit B	U.S. Patent No. 8,532,286 B2 to Robert J. Lambert (filed July 19, 2010, issued September 10, 2013)
Exhibit C	U.S. Patent No. 7,372,960 B2 to Robert J. Lambert (filed January 29, 2002, issued May 13, 2008)
Exhibit D	U.S. Patent No. 8,666,062 B2 to Robert J. Lambert (filed April 11, 2008, issued March 4, 2014)
Exhibit E	U.S. Patent No. 8,788,827 B2 to Marinus Struik et al. (filed September 14, 2012, issued July 22, 2014)
Exhibit G	U.S. Patent No. 7,372,961 B2 to Scott A. Vanstone et al. (filed December 26, 2001, issued May 13, 2008)
Exhibit M	U.S. Patent No. 5,159,632 (issued to Richard E. Crandall)
Exhibit N	US Patent Pub. No. US 2002/0122555 (issued as U.S. Patent No 6,751,318 to Richard E. Crandall)
Exhibit O	Jorge Guajardo et al, *Efficient Software-Implementation of Finite Fields with Applications to Cryptography* (2006)
Exhibit P	National Institute of Standards and Technology, *Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186* (1994)
Exhibit R	Alfred J. Menezes et al, *Handbook of Applied Cryptography* (1996)
Exhibit S	Pierre L'Ecuyer, *Uniform Random Number Generation* (1994)
Exhibit T	Peter L. Montgomery, *Modular Multiplication Without Trial Division* (1985)
Exhibit U	Dorothy Elizabeth Robling Denning, *Cryptography and Data Security* (1982)
Exhibit V	Darrell Hankerson et al, *Guide to Elliptic Curve Cryptography* (2004)
Exhibit W	*The Facts on File Dictionary of Computer Science* (2001), page 221
Exhibit X	*Microsoft Computer Dictionary, 5th ed.* (2002), page 573
Exhibit Y	Richard P. Brent et al, *Modern Computer Arithmetic Version 0.2* (2008)

II. RELEVANT LEGAL STANDARDS

17. I am not an attorney, so my understanding of patent law is based on explanations provided to me by counsel. The legal standards set forth below are based on the guidance provided by counsel, and I have applied these standards to my analysis.

18. I have been informed that patent claim terms should be interpreted consistent with the understanding that a person of ordinary skill in the art ("POSITA") would have had at the time of the alleged invention. I have been informed that the POSITA is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification. I have been informed that the prosecution history can often inform the meaning of the claim language by demonstrating how the inventor understood the invention and whether the inventor limited the invention in the course of prosecution, making the claim scope narrower than it would otherwise be.

19. I have been informed by counsel that Courts may also consider "extrinsic evidence" (e.g., inventor testimony, dictionaries, and treatises) when the intrinsic record alone is insufficient to support proper constructions. I further understand that expert testimony can be helpful to illuminate complex technical issues and provide a foundation for the viewpoint of one of ordinary skill in the relevant art.

20. I have been informed that a patent claim is invalid for indefiniteness if the claim, read in light of the specification and the prosecution history, fails to inform, with reasonable certainty, those skilled in the art about the scope of the alleged claimed invention. It is my understanding that indefiniteness is to be evaluated from the perspective of one skilled in the art at the time of the filing of the patent or the alleged priority date. I further understand that, although courts have recognized that some modicum of uncertainty is acceptable, a patent must be precise enough to afford clear notice of what is claimed and the scope of the invention.

III. LEVEL OF ONE OF ORDINARY SKILL IN THE RELEVANT ART

21. For purposes of formulating the opinions in this declaration, I have been asked by counsel to assume that each of the patents is entitled to the following earliest priority date:

- ’286 Patent: July 17, 2009
- ’960 Patent: December 31, 2001
- ’062 Patent: December 31, 2001
- ’827 Patent: January 18, 2005
- ’370 Patent: January 18, 2005
- ’961 Patent: December 27, 2000

22. Based on my experience in the field and my review of the patent, I believe that a POSITA relevant to the Asserted Patents as of the respective earliest priority dates would have at least a bachelor’s degree in computer science, mathematics, applied mathematics, or a related field, and (1) two or more years of experience in applied cryptography or computer security software engineering, or (2) an advanced degree in computer science or a related field.

23. Based on my education, training, and experience, it is my opinion that I can accurately represent the views of a POSITA as of the Asserted Patents’ respective earliest priority dates. Thus, I provide the opinions in this declaration using the viewpoint of a POSITA as of the respective earliest priority dates for each Asserted Patent.

IV. TECHNICAL BACKGROUND

A. Modular Arithmetic and Reduction

24. Modular arithmetic is a type of arithmetic performed on integers where the numbers “wrap around” a certain number called the **modulus**, often denoted as n. One analogy is how hours work on a standard clock having 12 hours. If it is 9 o’clock now and 5 hours pass, the hour

hand points to 2 o'clock instead of 14. In ordinary arithmetic, adding 5 to 9 yields 14; but for a clock, adding 5 hours to 9 o'clock results in 2 on the clock face because the numbers "wrap around" 12. This analogy is equivalent to performing modular arithmetic "mod 12", i.e., in a system having the modulus 12.

1. Modular Reduction

25. Given a modulus *n*, the integers [0, 1, 2…*n*–1] are called the canonical residues of *n*. Ex. R at 525-26. An integer *a* that is not within [0, 1, 2…*n*–1] may be "reduced" back to a canonical residue by dividing *a* by the modulus and taking the remainder. *Id.* Such an operation is referred to as the *modulo* operation, commonly denoted as *a* mod *n*. *Id.* For example, 7 mod 2 = 1 and 10 mod 4 = 2. The mod operation is often referred to as "modular reduction" or "reduction," because it "reduces" the value of a number larger than the modulus to below the modulus. *See* Ex. B at 1:20-23 ("In cryptography, e.g., public key cryptography, operation such as multiplication or exponentiation of integers in some group Zn may be required, where modular arithmetic is used to operate on the integers."), 1:29-30 ("The calculation of the remainder is referred to as reduction in modular arithmetic.").

26. The "≡" sign denotes "congruent to" or "modularly equivalent to," meaning two numbers are equivalent given a certain modulus. Ex. R at 525. For example, in the analogy of the clock above, we can write the operation of adding 5 to 9 in a system of mod 12 as: $9 + 5 \equiv 2 \pmod{12}$. The formula means 9 + 5 (14) is congruent to 2 in the system having the modulus 12. More formally, two numbers are congruent mod *n* (i.e., modularly equivalent) if they have the same remainder when divided by *n*. For example, 38, 26, 14, and 2 are all congruent to each other mod 12 because each of them, when divided by 12, has a remainder of 2. It should be clear that, when the difference between two numbers is an integer multiple of the modulus *n*, the two numbers are

congruent to each other mod *n*. *Id.*at 525-526. Two numbers congruent to each other mod *n* are interchangeable in the mod *n* system. *Id.*

27. Negative integers can also be reduced mod *n* by adding to it an integer multiple of *n* such that the result falls between 0 and $n - 1$. Some examples are shown below:

$$-2 \bmod 4 = -2 + 4 = 2$$

$$-9 \bmod 5 = -9 + (2 \times 5) = 1$$

2. Modular Arithmetic

28. I will now explain how modular arithmetic is performed. In general, the goal of modular arithmetic is to compute *a* op *b* mod *n*, where "*op*" denotes an arithmetic operation and *n* is the modulus. Ex. U at 72. For example, finite field addition calculates $a + b$ mod *n*; finite field multiplication calculates $a \times b$ mod *n*. Ex. R at 1054-1055; Ex. V at 1682.

29. Modular addition, subtraction, and multiplication can be performed simply by first performing ordinary addition, subtraction, or multiplication, followed by modular reduction. Ex. R at 526; Ex. V at 1682. For example:

$$4 + 5 \bmod 7 = 9 \bmod 7 = 2$$

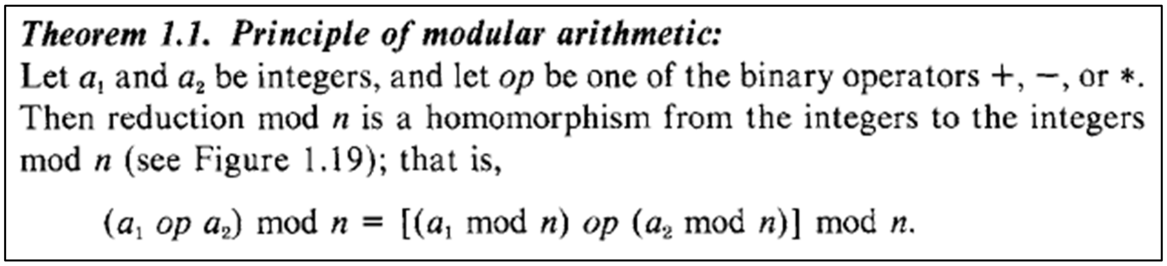
$$4 - 5 \bmod 7 = -1 \bmod 7 = 6$$

$$4 \times 5 \bmod 7 = 20 \bmod 7 = 6$$

30. Modular inversion and division are less similar to ordinary inversion and division because there are no fractions in modular arithmetic. In ordinary arithmetic, the inversion of 10 is $\frac{1}{10}$ which is 0.1. In other words, the goal of the ordinary inversion operation is to, given an input *a*, find an output *b* such that $a \times b = 1$. Similarly, in modular arithmetic, the inversion of a number *a* is the operation finding a number *b* such that $a \times b$ mod $n = 1$. Ex. R at 526. For example, assume a modulus $n = 7$, the modular inversion of 3 is 5 because $3 \times 5 \bmod 7 = 1$. *Id.* The

modular inversion of a is denoted as a^{-1} mod n. *Id.* Modular division is equivalent to multiplying the dividend with the inversion of the divisor. *Id.* In other words, $a \div b$ mod $n = a \times b^{-1}$ mod n. *Id.* Note that a modular inverse only exists when the operand and the modulus are coprime, meaning they have a greatest common divisor of 1. Ex. U at 76 ("Theorem 1.2").

31. One fundamental theorem of modular arithmetic is that, for addition, subtraction, and multiplication, "reducing each intermediate result" with the field modulus n "gives the same answer as computing in ordinary integer arithmetic and reducing the result mod n." *Id.* at 72-73.



Theorem 1.1. Principle of modular arithmetic:
Let a_1 and a_2 be integers, and let *op* be one of the binary operators $+$, $-$, or $*$. Then reduction mod n is a homomorphism from the integers to the integers mod n (see Figure 1.19); that is,

$$(a_1 \; op \; a_2) \bmod n = [(a_1 \bmod n) \; op \; (a_2 \bmod n)] \bmod n.$$

***Id.* at 73**

32. For example, one may calculate 25 multiplied by 15 with a modulus of 97 in the following way: $25 \times 15 \; mod \; 97 = 375 \; mod \; 97 = 84$. One may alternatively perform the following calculations based on Theorem 1.1 above to (1) obtain partial products, (2) reduce each partial product, and (3) accumulate the reduced partial product, which yields the same result:

$$\begin{aligned}
&(25 \times 15) \; mod \; 97 \\
&= [(20 + 5) \times (10 + 5)] \; mod \; 97 \\
&= [200 + 100 + 50 + 25] \; mod \; 97 \\
&= [200 \; mod \; 97 + 100 \; mod \; 97 + 50 \; mod \; 97 + 25 \; mod \; 97] \; mod \; 97 \\
&= [6 + 3 + 50 + 25] \; mod \; 97 \\
&= 84
\end{aligned}$$

33. The latter method, although appearing to involve more steps on paper, can provide advantages in computer-implemented cryptography which perform calculations in binary numbers. As shown in the comparative examples above, the first method (without intermediate reduction) involves an intermediate result of 375, which is 101110111 (9 bits) in binary

representation. In the second method, the largest intermediate result is 200, which is 11001000 (8 bits) in binary representation. The latter method, therefore, reduces the required storage space by 1 bit. The difference in required storage space would be more significant in actual implementations of cryptography, which typically use long finite field elements. *See* Ex. D at 2:43-45 ("Since the finite field used in ECC operations are typically 160 bits or more…").

B. Finite Field and Computer Implementations

34. Modular arithmetic is particularly useful in cryptography because many cryptographic systems rely on finite fields. A finite field is a field[1] having a finite number of elements. Ex. R at 538. One type of finite field widely used in cryptography is called a "prime field" (commonly denoted "F_p"), which is a field having a prime number of field elements that are consecutive integers starting from 0. *Id.* at 539. For example, a prime field having 97 elements consists of field elements integers 0 to 96. For such a finite field, the number of elements is defined as the modulus of the field. Performing an operation on elements of the finite field should yield an output that is also within the field. *See* Ex. U at 85. In other words, output of arithmetic in a finite field should "wrap around" the field modulus. *Id.* Therefore, modular arithmetic is used to perform arithmetic in finite fields.

35. For purposes of this declaration, I will focus on two types of finite fields – prime fields as discussed above, and binary extension fields (sometimes referred to as binary field, denoted by F_{2^n}). Ex. D at 2:27-30; Ex. U at 85. The elements of a binary field can be represented as polynomials with binary coefficients (i.e., 1 or 0). *Id.* For example, the elements of a binary field F_{2^4} are polynomials $a_3x^3 + a_2x^2 + a_1x + a_0$, where each of $a3$, $a2$, $a1$, $a0$ has the value 0 or 1.

[1] A field is a set of numbers with the usual operations of addition, multiplication, and division and all the usual algebra rules hold.

For convenience, the elements of this binary field are represented by a vector ($a3$, $a2$, $a1$, $a0$) of length 4. *Id.* For example, 0010 and 0100 represent elements of the binary field F_{2^4}. The modulus of a binary field is an irreducible polynomial $f(x)$, which is defined as a polynomial that cannot be factored into smaller polynomials. Ex. R at 569.

36. In computers, finite field moduli and elements are represented in binary. Ex. D at 2:27-31. For prime fields, the binary representation is simply the binary form of the integer modulus or field element. For binary fields, the binary representation is the vector representation discussed above. Computer implementation of finite-field-based cryptography typically requires a large finite field. *Id.* at 2:43-44 ("finite field used in ECC operations are typically 160 bits or more."). To facilitate easier and faster computation, finite field elements are typically broken into and stored in multiple machine "words." *Id.* at 2:44-45 ("[Finite field] elements must be represented in several machine words.").

37. Each computer processor has its native "word" size, which is a number of binary bits the processor can handle "as a single unit." Ex. W. The word-size typically corresponds to the processor's register width, and is commonly indicated in the "n-bit" designation of the processor. *Id.*; Ex. V at 1680. For example, a "32-bit" processor would have a 32-bit word size. A processor can perform word-sized operations most efficiently, such as adding two 32-bit integers on a 32-bit processor. Ex. X; *see* Ex. T at 1275. Performing the same type of operation on data larger than the native word size typically takes much longer because it requires multiple instructions. *Id.*

38. Processors typically have word size of 64-bit or less. Computer implementations of cryptographic systems typically use finite fields having elements much longer than the processor's word size, and therefore "finite field elements are often too long to be represented in

a single machine word…[and] these elements must be represented in several machine words." Ex. D at 2:39-45; *see e.g.*, Ex. V at 1695 (disclosing cryptosystems using finite field elements up to 521-bit long).

39. A common way to store field elements in words is to divide the bit-length of the field modulus with the word size of the system, then round up to an integer. Ex. V at 1680. This is the number of words required to store a finite field element, since all elements are smaller than the modulus. For example, assume we are working on a 32-bit processor with a prime field F_{521} whose modulus is $2^{521} - 1$ (this is a prime number). The length of the modulus is 521 bits and the word size of 32 bits. Therefore, 17 words (521 / 32, rounded up) are required to store a field element a. A common way to denote the word-sized representation of a is $a = (a_{16}, \ldots a_1, a_0)$. *See id.*

40. In the example above, the right-most word (a_0) is called the "least significant word" ("LSW") while the left-most word (a_{16}) is called the "most significant word" ("MSW"). *See e.g.*, Ex. B at 5:4-6. The names come from the fact that the right-most word a_0 makes the least contribution of the value of the a while the left-most word a_{16} makes the most contribution. As an example in decimal, assume a word size of 3, then the number 123,456,789 broken into words is a_2 = 123 (the MSW), a_1 = 456, and a_0 = 789 (the LSW).

C. Methods for Performing Modular Reduction in Cryptography

41. As explained above, performing finite field arithmetic involves modular reduction, which involves dividing an input with the modulus and calculating the remainder. However, performing division on a computer can be computationally difficult and expensive. Therefore, many optimized methods for performing modular reduction and calculate x mod n, where x is typically the result of a preceding arithmetic operation, have been developed. I explain two below that are relevant to the disputed claims.

1. Montgomery Reduction

42. Montgomery reduction was originally proposed by Peter Montgomery in 1985. Ex. B at 1:32-34 ("Of the well known methods for modular reduction, the most commonly used is the method of Montgomery modular reduction."), 2:14-17 ("The use of Montgomery reduction as a component of Montgomery multiplication is well known. There are many algorithms that can be used to perform the Montgomery multiplication."); Ex. T.

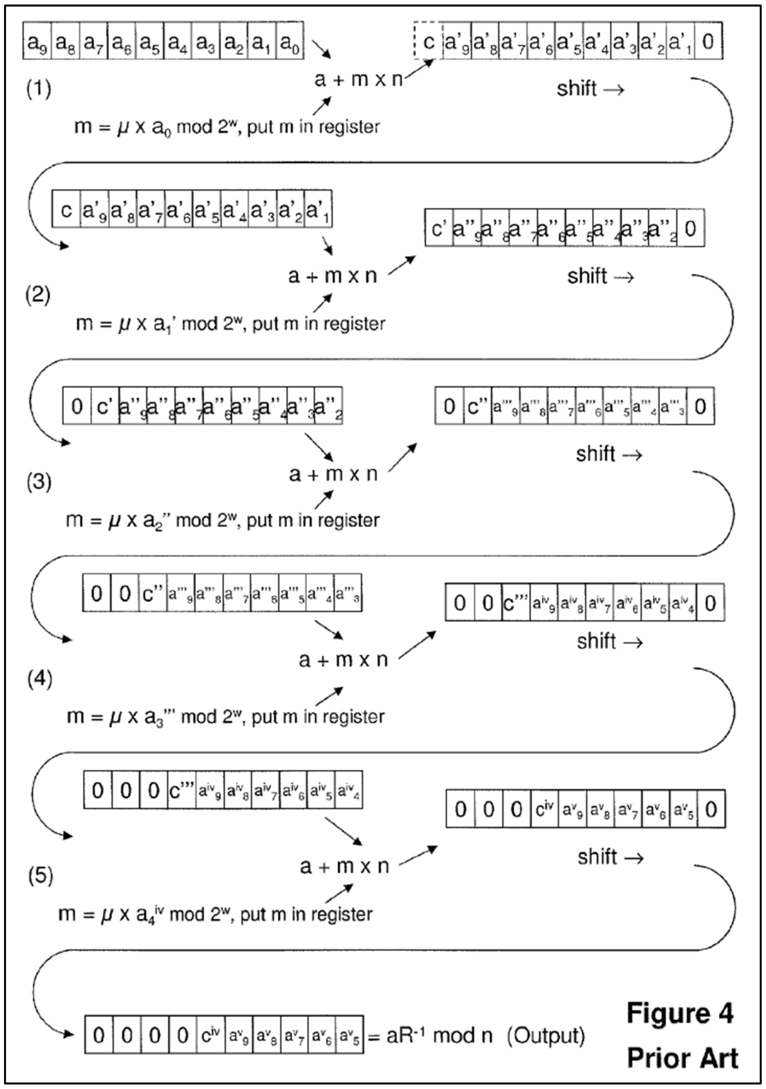
43. Montgomery reduction and related arithmetic uses a special radix R, which is typically a large power of 2 (i.e., 2^k). Ex. B at 1:47-52; Ex. T at 1275. When performing modular arithmetic using the Montgomery method, an operand a is first converted to its Montgomery form, defined as aR mod n. Ex. T at 1276. Operations are then carried out in the Montgomery form. *Id.* For example, to calculate $a \times b$ mod n, one first converts a and b into their Montgomery form: aR mod n and bR mod n and then calculates aR mod $n \times bR$ mod $n = abR^2$ mod n. The Montgomery reduction function is then applied to the product abR^2 mod n twice to remove the R terms and convert back the result from its Montgomery form. *Id.* Specifically. the Montgomery reduction function (commonly denoted as the **REDC()** function) takes as input a number T (typically, the result of a preceding arithmetic operation) and outputs TR^{-1} mod n. *Id.* at 1275. In the example of multiplication above, applying Montgomery reduction twice to the product in Montgomery form abR^2 performs: REDC(abR^2 mod n) = abR mod n; REDC(abR mod n) = ab mod n.

44. Performing Montgomery reduction involves first adding to the reductant a carefully chosen integer multiple of the modulus (i.e., $T + m \times n$) such that the sum becomes divisible by the radix $R = 2^k$. *Id*. at 1275-1276. In other words, adding the integer multiple of modulus $m \times n$ to the reductant zeroes the least significant k bits of the reductant. One then divide the modified

reductant by R. Note that adding $m \times n$ does not change the congruence of the reductant (i.e., $T + m \times n \equiv T \bmod \text{n}$). This ensures that the result is in fact $TR^{-1} \bmod n$.

45. Thus, a defining characteristic of Montgomery reduction is that it reduces "from below," meaning from the least significant portion of the reductant. In contrast, many other methods for modular reduction reduce "from above" (i.e., from the most significant portion of the reductant), such as Euclidean division and pseudo-Mersenne reduction, to be discussed below. *See* Ex. B at 1:41-46 ("[A]s opposed to classical methods of reduction-from-above such as Euclidean division, Montgomery reduction reduces from below, that is, the method proceeds by clearing the least-significant portions of the unreduced quantity, leaving the remainder in the upper portion.").

46. Montgomery reduction may also be performed word-by-word, where adding the integer multiple only zeros the LSW of the reductant. The process is repeated once per word until all words corresponding to the k bits of R have been processed. The '286 Patent describes such a prior art method for Montgomery reduction, using a typically pre-computed value $\mu = (-n)^{-1} \bmod 2^w$ to find the integer multiple, where w is the word-size of the system and n is the modulus. *Id*. at 2:24-25. The '286 Patent illustrates an example of this process in Figure 4 and at 5:24-25 ("As shown in FIGS. 3 and 4, a typical Montgomery-style reduction…"). *Id.* As shown below, the input a consists of 10 words (a_9, … a_0). The prior art method teaches adding $m \times n$ to a, where $m = \mu \times a \bmod 2^w$. After the addition, the LSW of a becomes zero. One then shift a to the right by one word, thus reducing the length of a. The process is repeated a number of times until the desired amount of reduction is achieved to produce an output that is $aR^{-1} \bmod n$.



(1)
m = μ x a0 mod 2^w, put m in register
a + m x n
shift →
(2)
m = μ x a1' mod 2^w, put m in register
(3)
m = μ x a2'' mod 2^w, put m in register
(4)
m = μ x a3''' mod 2^w, put m in register
(5)
m = μ x a4^iv mod 2^w, put m in register
= aR^-1 mod n (Output)
Figure 4
Prior Art

***Id.* at Fig. 4**

47. An example in decimal is provided below to illustrate how Montgomery reduction is used to calculate **4355 mod 97 = 87**.

Montgomery Reduction

Assumptions: modulus n = 97; word-size = 2; a = 4355; Montgomery radix $R = 10000$

Goal: calculate 4355 mod 97 (i.e., a mod n)

Answer: 4355 mod 97 = 87

1. **Convert a to Montgomery form (T):**

   $T = a \times R \bmod n = 4355 \times 10000 \bmod 97 \equiv 39195$

2. **Calculate μ and m:**

   $\mu = (-n)^{-1} \bmod 10^{w} = (-97)^{-1} \bmod 100 = 67$

   $m = \mu \times T_0 \bmod 10^{w} = 67 \times 95 \bmod 100 = 65$

3. **Add m × n to T such that the LSW (the last two digits) is zero:**

   $39195 + 65 \times 97 = 45500$

4. **Shift T to the right by 2 digits:**

   $45500 \rightarrow 455$

5. **Calculate m**

   $m = \mu \times T_0 \bmod 10^{w} = 67 \times 55 \bmod 100 = 85$

6. **Add m × n to T such that the LSW (the last two digits) is zero:**

   $455 + 85 \times 97 = 8700$

7. **Shift T to the right by 2 digits:**

   $8700 \rightarrow 87$

2. Pseudo-Mersenne Reduction

48. Another modular reduction method commonly used in cryptography is pseudo-Mersenne reduction, first proposed by Richard Crandall in U.S. Patent No. 5,159,632 issued in 1992. *See* Ex. N (continuation publication of U.S. Pat. No. 5,159,632). The method became widely known and used. For example, a 2006 publication, Guajardo, describes the method. Ex. O at 3599 (citing Ex. M), 3600 (describing algorithm for "Fast reduction modulo a pseudo-Mersenne prime"); *see also* Ex. R at 1060 (1996 textbook describing pseudo-Mersenne reduction). This reduction method requires using a special type of modulus called a pseudo-Mersenne number, which is typically a prime number having the form $n = 2^k - c$, where k is an integer and c is another integer much smaller than 2^k. *Id.* One example of a pseudo-Mersenne number is $2^{255} - 19$ (i.e., k

= 255, c = 19). Note that for calculations in decimal, such as the example in the following paragraph, the special form of the modulus is $10^k - c$.

49. Using a pseudo-Mersenne prime modulus simplifies calculations because 2^k mod n = c (or in decimal calculations 10^k mod n = c). *See* Ex. O at 3599. Therefore, calculating an integer multiple of 2^k mod n (i.e., $a \times 2^k$ mod n) can be replaced with the much simpler calculation of $a \times c$. An example in decimal is provided below to illustrate how the method is used to calculate **4355 mod 97 = 87**.

Pseudo-Mersenne Reduction
Assumptions: modulus n = 97; word-size = 2; a = 4355
Goal: calculate 4355 mod 97 (i.e., a mod n)
Answer: 4355 mod 97 = 87
1. Obtain c, where n = 10^k – c
$97 = 10^2 - 3$
c = 3
2. Split a = 4355 into a high word (h) and a low word (l):
h = 43; l = 55
3. Calculate a mod n = h × c + l:
43 × 3 + 55 = 184
4. Split t = 184 into a high word (h) and a low word (l):
h = 1; l = 84
5. Calculate t mod n = h × c + l:
1 × 3 + 84 = 87

50. Note that the pseudo-Mersenne reduction method is different from Montgomery reduction in that it reduces "from above," in other words from the more significant bits. In the decimal example above, the reductant 4355 = 43 × 100 + 55. In the step where t is split into a high word and a low word, the multiplicand of 100 is dropped, and is replaced with a much smaller number of c (3). This step amounts to reducing the more significant words of the operand—*i.e.*, reducing from above.

V. THE PATENTS-IN-SUIT

A. The ’286 Patent

51. As explained above, Montgomery reduction involves adding a multiple of the modulus ($m \times n$) to a reductant T, such that the least significant word of the sum (i.e., $T + m \times n$) is always zero. A POSITA would understand that this step is called “cancellation,” which is a term understood in the art of modular arithmetic of performing an operation that “zeros” certain words. *See, e.g.*, Ex. Y at 3440-3441 (“This cancellation can be dramatic, as in 6.7782357934 – 6.7782298731 = 0.0000059203.”). In Montgomery-style reduction, a POSITA would have understood said operation to be the addition of a multiple of the modulus and that that addition cancels or zeros the LSW. The sum is then shifted to the right or down to drop the zeroed word. The process then repeats on the shifted value until a desired number of words are reduced. Section IV.C.1, *supra*.

52. The ’286 Patent is directed to a modified version of the prior art Montgomery reduction. *See, e.g.*, Ex. B at 1:13-16 (“The following…provides a system and method for reducing the computation and storage requirements for a Montgomery-style reduction.”). The ’286 Patent’s method “produce[s] a Montgomery reduction…by storing a new precomputed value used to substantially replace the μ and n values used in Montgomery reduction with a single value.” *Id.* at 3:27-31. Specifically, the ’286 Patent describes using a “new value” $\mathbf{n' = 2^{-w} \bmod n}$ for performing Montgomery reduction. *Id.* at 5:45-59 (stating that *n’* is “used to zero the least significant non-zero word of a at each iteration, without the need to first multiply by μ and determine m.”). The ’286 Patent then teaches performing a replacement of the operand “$a \equiv [\ldots, a_4, a_3, a_2, a_1, a_0]$…with $a \equiv [\ldots, a_4, a_3, a_2, a_1, 0] + a_0 \times n' \times 2^w$.” *Id.* at 5:60-6:2. It describes this replacement step as zeroing the LSW a_0 and adding the term “$a_0 \times n' \times 2^w$,” which is necessarily modularly

equivalent to a_0 mod n. *Id.* The LSW can then be dropped by shifting the entire result to the right or down. *Id.*

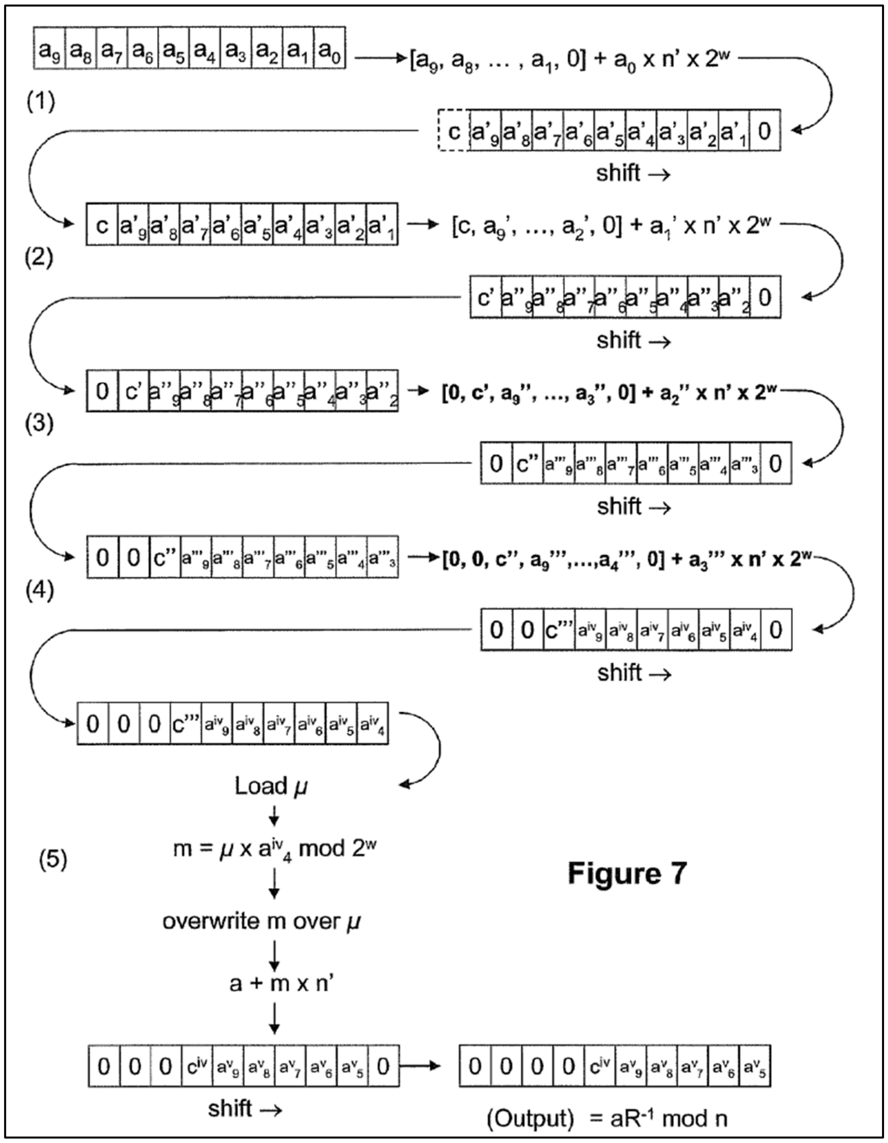
53. The term "replace" or "replacement" in the context of modular arithmetic was not a term of art prior to the '286 Patent and a POSITA would not have understood what it means without referring to the specification of the '286 Patent. In contrast, a POSITA would have understood that the term "cancellation" in this context refers to the step in standard Montgomery reduction wherein a multiple of the modulus is added such that the resulting sum always has an LSW of zero, and the entire result can be shifted right or down to drop the LSW.

54. The inventors of the '286 Patent therefore used the term "replacement" to differentiate the purportedly inventive step used in the '286 patent's method from the well-known technique of cancellation used in standard Montgomery reduction. *See, e.g.*, *id.* at 3:27-39 ("In the following embodiments, a system and method are utilized that provide ***an alternative way in which to produce a Montgomery reduction from below*** by storing a new precomputed value used to substantially replace the μ and n values used in Montgomery reduction with a single value. This may be done by storing a modified reduction value in the cryptographic apparatus, wherein the modified reduction value … ***performs a replacement for values in a low-order segment which is a target of the reduction, rather than a cancellation thereof, as performed in a standard Montgomery reduction***; and performing the reduction from below using the modified reduction value."). I understand that MARA's construction of the "replacement" term is "add a modular equivalent of the operand's least significant word to the more significant words of the operand such that the result can be shifted down to drop the least significant word." This construction is consistent with, but not limited to, the sole embodiment in the specification, which explicitly teaches a POSA what replacement means. *Id.* at 5:59-67 ("To be explicit a=[...,a4,a3,a2,a1,a0] is

replaced with $a=[...,a4,a3,a2,a1,0] + a_0 \times n' \times 2^w$. Since $a_0 \times n' \times 2^w$, taken without reduction, is zero in its least significant digit (by the shift 2^w), the resulting value is 0 in its least significant digit, which is the desired low-order reduction. Typically this zero digit will be treated by shifting (either logically or physically) the value down by a digit.").

55. This embodiment of the '286 Patent's modified method for Montgomery reduction is illustrated in Figure 7. *Id.* at 6:32-35 ("As shown in FIG. 7, by obtaining and using the modified reduction value n′ and applying the relationship $a \equiv [. . . ,a_4,a_3,a_2,a_1,0]+a_0 \times n' \times 2^w$ at each iteration, the least significant word of a is zeroed and the remaining words modified."). As shown below, the input *a* consists of 10 words (a_9, … a_0). The '286 Patent teaches setting a_0 (LSW of a) to zero and adding $a_0 \times n' \times 2^w$ to a, where $n' = 2^{-w} \bmod n$.[2] Because the LSW of the addend $a_0 \times n' \times 2^w$ is also zero (multiplying 2^w adds *w* number of zeros to the end of the addend), the LSW of post-addition *a* is now zero. *Id.* at 5:60-6:2 ("$a_0 \times n' \times 2^w$, taken without reduction, is zero in its least significant digit."). One may then shift *a* to the right by one word to take the space of the a_0, thus reducing the length of *a*. The process is repeated a number of times until the desired amount of reduction is achieved to produce an output that is $aR^{-1} \bmod n$. Note that the '286 Patent discloses that its method cannot be used for the last iteration, and instead the standard prior art Montgomery method must be used. *Id.* at 6:44-50, 9:1-2 ("Now using standard Montgomery reduction for the last digit.").

[2] Note that these two steps can be performed in either order without affecting the result.


(1)
[a9, a8, … , a1, 0] + a0 x n' x 2w
shift →
(2)
[c, a9', …, a2', 0] + a1' x n' x 2w
shift →
(3)
[0, c', a9'', …, a3'', 0] + a2'' x n' x 2w
shift →
(4)
[0, 0, c'', a9''',…,a4''', 0] + a3''' x n' x 2w
shift →
(5)
Load μ
m = μ x aiv4 mod 2w
overwrite m over μ
a + m x n'
shift →
(Output) = aR-1 mod n
Figure 7

***Id.* at Fig. 7**

56. Below, I provide an example in decimal to illustrate how the '286 Patent's method is used to calculate 4355 mod 97. Note that, in the example below, the radix is 10000 (i.e., 10^4) instead of 2^4. This is because the example is performed in decimal, where the base is 10 instead of 2.

’286 Patent Method (Fig. 7)

Assumptions: modulus n = 97; word-size = 2; a = 4355

Goal: calculate 4355 mod 97 (i.e., a mod n)

Answer: 4355 mod 97 = 87

1. **Convert t to Montgomery form (T):**

   $T = a \times R \bmod n = 4355 \times 10000 \bmod 97 \equiv 39195$

2. **Calculate n’ = 10^{-w} mod n:**

   $n' = 10^{-2} \bmod 97$

   $n' = 65$

3. **Zero the LSW of T**

   $39195 \rightarrow 39100$

4. **Add $T_0 \times n' \times 10^w$ to T:**

   $39100 + 95 \times 65 \times 10^2 = 656600$

5. **Shift T to the right by two digits:**

   $656600 \rightarrow 6566$

Perform standard Montgomery reduction on 6566:

6. **Calculate μ and m:**

   $\mu = (-n)^{-1} \bmod 10^w = (-97)^{-1} \bmod 100 = 67$

   $m = \mu \times T_0 \bmod 10^w = 67 \times 66 \bmod 100 = 22$

7. **Add m × n to T such that the last two digits are zero:**

   $6566 + 22 \times 97 = 8700$

8. **Shift T to the right by 2 digits:**

   $8700 \rightarrow 87$

57. While both the standard Montgomery method and the ’286 Patent’s method involve clearing the least significant portions of an unreduced operand and leaving the remainder in the more significant portions, each does so in mathematically distinct ways. Indeed, the pre-shift and shifted results from both techniques are modularly equivalent to one another and are therefore interchangeable and ultimately result in the correct calculation of 4355 mod 97.

<u>Pre-Shift:</u>
656600 mod 97 = 7 (’286 Method)
45500 mod 97 = 7 (Montgomery Method)

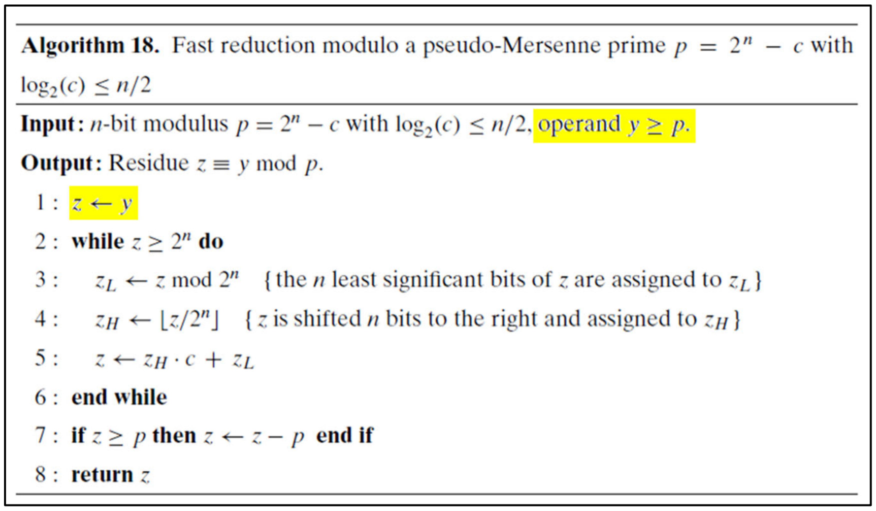
<u>Shifted:</u>
6566 mod 97 = 67 ('286 Method)
455 mod 97 = 67 (Montgomery Method))

58. I have been informed that Malikie has proposed the following constructions of claim 1 of the '286 Patent: (1) the preamble including "Montgomery-style reduction" is not limiting; (2) "perform a replacement of a least significant word" means "replace a word that makes the smallest contribution to the value of the operand"; and (3) "perform a cancellation thereof" means "add a multiple of the modulus to the operand to eliminate the least significant word of the operand." As shown below in paragraphs 58-62, under these constructions proposed by Malikie, claim 1 would clearly cover the prior art pseudo-Mersenne reduction.

59. Jorge Guajardo et al., *Efficient Software-Implementation of Finite Fields with Applications to Cryptography* ("Guajardo") is a 2006 publication from Springer Science that discloses pseudo-Mersenne reduction. Ex. O.

60. As noted above, Malikie contends that the preamble of claim 1 ("A method for performing, on a cryptographic apparatus, a Montgomery-style reduction in a cryptographic operation, the method comprising:") is not limiting.

61. The first limitation of claim 1 of the '286 Patent is "obtaining an operand for the cryptographic operation." Guajardo discloses a reduction method (Algorithm 18) that is "an integral part of both polynomial multiplication and polynomial squaring," which are cryptographic operations. *Id.* at 3599. Algorithm 18, as shown below, obtains a variable *z* to store the value of *y*. Either *y* or *z* in Step 1 below is the "operand."



Algorithm 18. Fast reduction modulo a pseudo-Mersenne prime $p = 2^n - c$ with $\log_2(c) \le n/2$

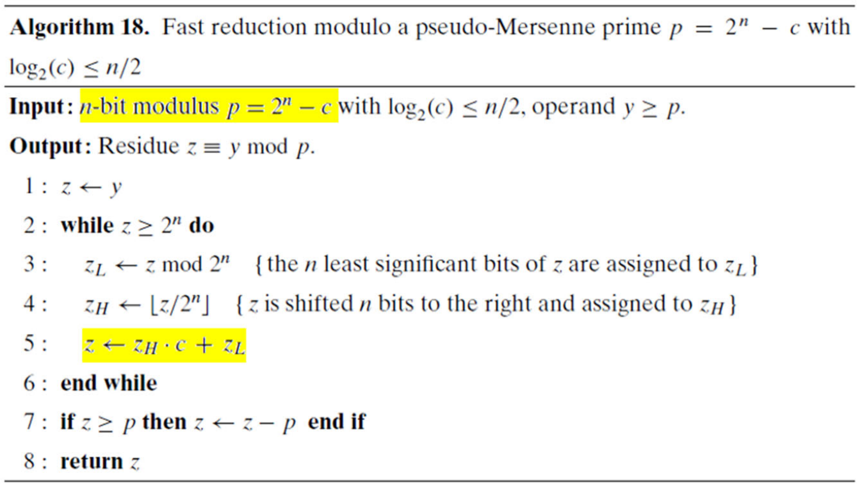
Input: n-bit modulus $p = 2^n - c$ with $\log_2(c) \le n/2$, operand $y \ge p$.
Output: Residue $z \equiv y \bmod p$.

```
1 : $z \leftarrow y$
2 : while $z \ge 2^n$ do
3 :     $z_L \leftarrow z \bmod 2^n$   {the $n$ least significant bits of $z$ are assigned to $z_L$}
4 :     $z_H \leftarrow \lfloor z/2^n \rfloor$   {$z$ is shifted $n$ bits to the right and assigned to $z_H$}
5 :     $z \leftarrow z_H \cdot c + z_L$
6 : end while
7 : if $z \ge p$ then $z \leftarrow z - p$ end if
8 : return $z$
```

***Id.* at 3600**

62. The next limitation of claim 1 requires "computing a modified operand using a reduction value, instead of a modulus used in performing a standard Montgomery reduction, to perform a replacement of a least significant word of the operand, rather than perform a cancellation thereof, the reduction value being a function of the modulus." First, as shown below, Algorithm 18 at step 5 uses a reduction value *c* to compute a modified operand. The modified operand is calculated by adding $z_H \cdot c$ to the z_L (which is the LSW of the operand). Second, the reduction value *c* is not a modulus used in performing a standard Montgomery reduction. The modulus used in performing standard Montgomery reduction is denoted as *p* in Algorithm 18. The reduction value *c* is a function of the modulus *p* (i.e., $c = 2^n - p$). Third, Guajardo uses the reduction value *c* to "perform a replacement of the least significant word of the operand" under Malikie's proposed interpretation, which merely requires that the least significant word of the operand be set. Algorithm 18 generates a modified operand in Step 5, and therefore, sets all words of modified operand including the least significant word. Fourth, Guajardo does not "perform a cancellation thereof," which Malikie construes as "add a multiple of the modulus to the operand to eliminate the least significant word of the operand." Guajardo does not add a multiple of the modulus *p* to

the operand, but instead, adds a multiple of the reduction value c to a portion of the operand. *Id.* at 3599 ("The basic reduction step is accomplished by multiplying z_H and c together and 'folding' the product $z_H \bullet c$ into z_L."). Fifth, as already noted above, the reduction value c is function of the modulus p. Guajardo teaches that $p = 2n - c$, and therefore, a POSITA would understand that $c = 2^n - p$.



Algorithm 18. Fast reduction modulo a pseudo-Mersenne prime $p = 2^n - c$ with $\log_2(c) \leq n/2$

Input: n-bit modulus $p = 2^n - c$ with $\log_2(c) \leq n/2$, operand $y \geq p$.

Output: Residue $z \equiv y \bmod p$.

```
1 : $z \leftarrow y$
2 : while $z \geq 2^n$ do
3 :     $z_L \leftarrow z \bmod 2^n$   {the $n$ least significant bits of $z$ are assigned to $z_L$}
4 :     $z_H \leftarrow \lfloor z/2^n \rfloor$   {$z$ is shifted $n$ bits to the right and assigned to $z_H$}
5 :     $z \leftarrow z_H \cdot c + z_L$
6 : end while
7 : if $z \geq p$ then $z \leftarrow z - p$ end if
8 : return $z$
```

***Id.* at 3600 (highlighting added)**

63. The last limitation of claim 1 is "outputting the modified operand." In Step 5, the value of $z_H \bullet c + z_L$ is stored in the z, which is then used in the next loop. *See also id.* at 3599 ("Repeating the substitution a few times and perform final subtraction of p yields the fully reduced result $x \bmod p$."). Thus, the modified operand z is output.

64. In my opinion, Malikie's proposed interpretation of claim 1 of the '286 Patent is clearly incorrect and not how a POSITA would understand claim 1 in light of the specification. The specification is entirely about a modification to standard Montgomery reduction, and not at all about the prior art pseudo-Mersenne reduction or any other type of reduction.

B. The '062 and '960 Patents

65. The asserted claims of the '062 and '960 Patents are directed to a method for performing a finite field operation using a processor, comprising the following steps: (1)

performing the finite field operation on representations of the finite field elements to generate an unreduced result completing the finite field operation, (2) reducing the unreduced result to generate a reduced result, and (3) providing the reduced result for use in a cryptographic operation. *See* Ex. D at claim 1; Ex. C at claim 3.

66. The patent describes that its invention is useful for cryptographic systems, including elliptic curve cryptography ("ECC"). Ex. D at 1:45-48 ("Elliptic curve cryptography (ECC) is a particularly efficient form of public key cryptography that is especially useful in constrained environments…"), 8:36-38 ("The finite field engine 400 provides finite field routines 430 for use by the cryptographic engine 200 and the elliptic curve engine 300."). The background section of the patent provides the following description of ECC and its relationship with finite field arithmetic:

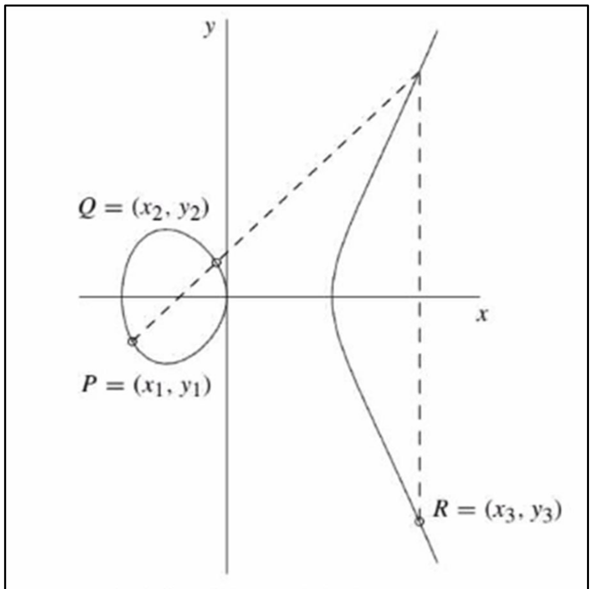
> To specify an elliptic curve, a finite field and an equation over that finite field are needed. The points on the elliptic curve are the pairs of finite field elements satisfying the equation of the curve, as well as a special point at infinity. To carry out calculations involving points on the elliptic curve, calculations are done in the underlying finite field, according to well-known formulas that use parameters of the curve.

Id. at 1:48-55.

67. Note that the specification of the patents makes clear that a finite field operation is different from an elliptic curve operation, which are operations on elliptic curve points. For example, the specification states that "elliptic curve consists of two finite field elements. The finite field elements are only operated on directly by the finite field engine." *Id.* at 7:36-41; *see also id.* at 7:20-32 (describing that a finite field engine comprises finite field operations), 6:52-54 (stating that implementing ECDSA protocol "requires the use of both elliptic curve operations and finite field operations."), 7:15-16 ("Each elliptic curve operation 320 requires certain finite field

operations."), 7:62-67 (states that elliptic curve operations "in turn direct finite field operations."), Figs. 3-4, 7 (listing elliptic curve operations and finite field operations separately).

68. Consistent with the patent specification, a POSITA would understand that elliptic curve operations operate on elliptic curves points, which each consist of two finite field elements as the *x*-coordinate and *y*-coordinate of the point. One example of elliptic curve operation is elliptic curve addition. *Id.* at 6:64-65 ("More specifically, pointer 222 references an elliptic curve addition operation."). The figure below illustrates how the addition of elliptic curve points *P* and *Q* is performed conceptually. First, one draws a line connecting *P* and *Q*. Second, the point at which the line intersects the curve is noted. Third, the reflection of point of intersection about the *x* axis, *R*, is the sum of points *P* and *Q*.

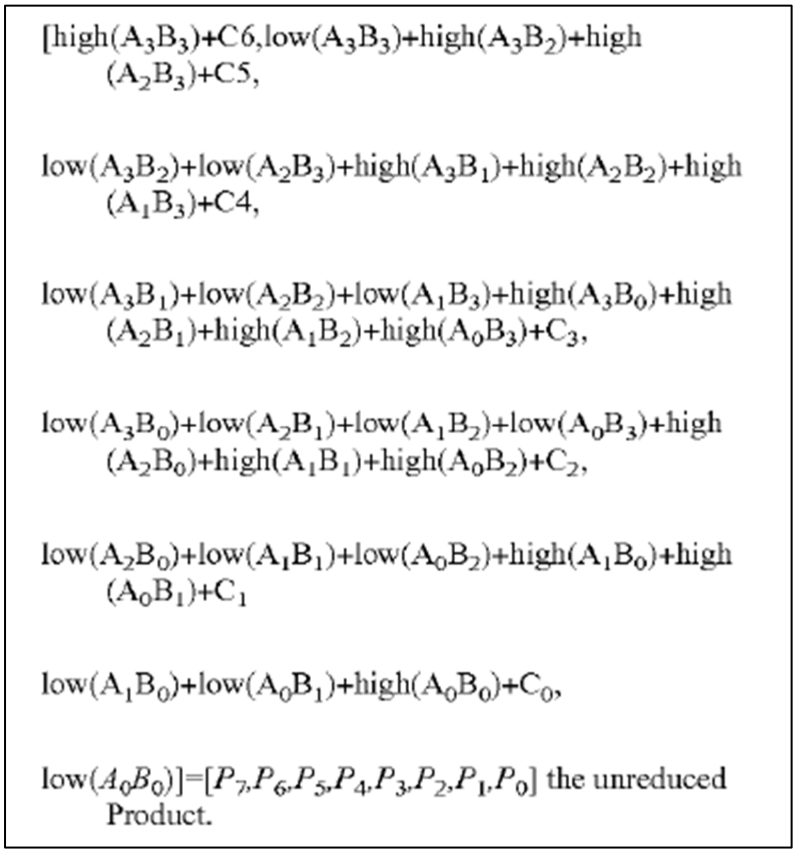


(a) Addition: $P + Q = R$.

Ex. V at 1731

69. As shown above, although elliptic curve addition has "addition" in its name, it is a completely different type of operation from finite field addition, which adds two finite field elements and reduces the result to the modulus. Section IV.A.2, *supra*.

70. The patents provide examples of performing wordsized finite field operations to generate an unreduced result. For example, performing finite field multiplication on two finite field elements A and B, each composed of four words, to generate an unreduced result. Ex. D at 11:45-65. As shown below, the process involves multiplying each word of A with each word of B to generated partial products, followed by accumulating the partial products to generate the unreduced product.



[high(A_3B_3)+C6,low(A_3B_3)+high(A_3B_2)+high (A_2B_3)+C5,

low(A_3B_2)+low(A_2B_3)+high(A_3B_1)+high(A_2B_2)+high (A_1B_3)+C4,

low(A_3B_1)+low(A_2B_2)+low(A_1B_3)+high(A_3B_0)+high (A_2B_1)+high(A_1B_2)+high(A_0B_3)+C_3,

low(A_3B_0)+low(A_2B_1)+low(A_1B_2)+low(A_0B_3)+high (A_2B_0)+high(A_1B_1)+high(A_0B_2)+C_2,

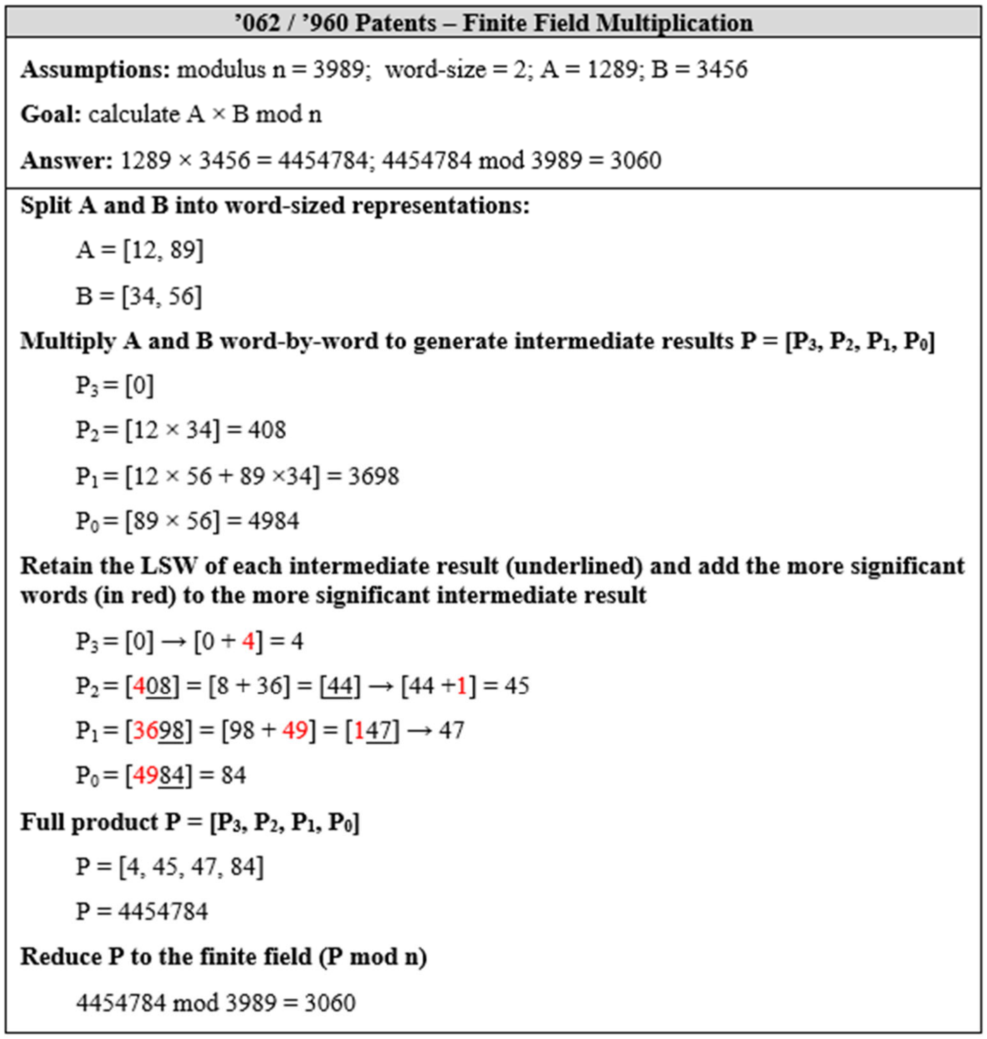
low(A_2B_0)+low(A_1B_1)+low(A_0B_2)+high(A_1B_0)+high (A_0B_1)+C_1

low(A_1B_0)+low(A_0B_1)+high(A_0B_0)+C_0,

low(A_0B_0)]=[P_7,P_6,P_5,P_4,P_3,P_2,P_1,P_0] the unreduced Product.

***Id.* at 11:47-65**

71. Below, I provide an example to illustrate how the multiplication method according to the patents is performed in decimal. Assume a word size of 2. Calculating A = 1289 and B = 3456 involves the following steps. Note that 1289 × 3456 = 4454784.



’062 / ’960 Patents – Finite Field Multiplication

Assumptions: modulus n = 3989; word-size = 2; A = 1289; B = 3456

Goal: calculate A × B mod n

Answer: 1289 × 3456 = 4454784; 4454784 mod 3989 = 3060

Split A and B into word-sized representations:

A = [12, 89]

B = [34, 56]

Multiply A and B word-by-word to generate intermediate results P = [P_3, P_2, P_1, P_0]

$P_3 = [0]$

$P_2 = [12 \times 34] = 408$

$P_1 = [12 \times 56 + 89 \times 34] = 3698$

$P_0 = [89 \times 56] = 4984$

Retain the LSW of each intermediate result (underlined) and add the more significant words (in red) to the more significant intermediate result

$P_3 = [0] \rightarrow [0 + 4] = 4$

$P_2 = [4\underline{08}] = [8 + 36] = [\underline{44}] \rightarrow [44 + 1] = 45$

$P_1 = [36\underline{98}] = [98 + 49] = [1\underline{47}] \rightarrow 47$

$P_0 = [49\underline{84}] = 84$

Full product P = [P_3, P_2, P_1, P_0]

P = [4, 45, 47, 84]

P = 4454784

Reduce P to the finite field (P mod n)

4454784 mod 3989 = 3060

72. The specification also discloses wordsized addition, subtraction, and inversion. *Id.* at 8:65-9:58 (addition), 9:59-64 (subtraction), 12:5-14:56 (inversion). All these methods are performed without intermediate reduction.

73. The specification describes two types of reduction: one that is “specific to a certain finite field, or a wordsize reduction.” *Id.* at 8:40-49. The first type of reduction, “specific to a certain finite field,” refers to reducing an input *a* to the modulus of the finite field *n* (i.e., calculating *a* mod *n*) and is what the claims cover. *Id.* at 14:59-60 (“The modular reduction routine is provided with instructions specific to the modulus used.”).

74. With respect to wordsize reduction, the specification explains that the finite field modulus and elements are typically stored in multiple words because they are generally larger than the wordsize of the system. *Id.* at 16:10-13. For example, the patents describe "[c]omputing the necessary number of machine words requires a simple calculation of the maximum field size needed divided by the machine word size, rounded up to an integer." *Id.* The patent then explains that the wordsize reduction "should lower the length of the result to the appropriate word length of the underlying field. This way, finite field elements may be consistently stored in registers of the same word length." *Id.* at 8:44-47. An example in decimal is provided below to illustrate the difference between reduction specific to a certain finite field and wordsize reduction. Assume a system having a wordsize of 3 and a finite field modulus of 10,007. The patents teach storing a finite field elements in two words because the modulus (i.e., 5 ÷ 3, rounded up).

- Reducing 10,000,000 with the specific finite field: 10,000,000 mod 10,007 = 3,007.
- Wordsized reduction of 10,000,000: generate a 6-digit (2-word) number congruent to 3,007 mod 10,007, for example 993,700.

75. I disagree with Malikie's proposed construction for the claim term "unreduced product," which is "result of an operation whose bit length has not been lowered." A POSITA would find Malikie's proposal vague and impossible to apply. It is unclear what the bit length of the unreduced result is compared against to determine whether it has been lowered. For example, in the wordsized multiplication example in the patents, two finite field elements are multiplied by generating 32 partial products followed by accumulating the partial products to generate an unreduced product. A POSITA would not be able to determine which entity the result should be compared with to determine that its bit length has not been lowered. *Id.* at 11:47-65.

76. I also disagree with Malikie's proposed construction for claim term "reduced result," which is "result of an operation whose bit length has been lowered." The claims explain

that the reduced result is obtained my performing modular reduction on the unreduced result. *Id.* at claim 1 ("obtaining a second set of instructions for performing a modular reduction for a specific finite field; executing the second set of instructions on the unreduced result to generate a reduced result."); Ex. C at claim 1 ("performing a specific modular reduction of said unreduced result to reduce said unreduced result to that of a field element of said finite field to obtain a reduced result."). However, performing modular reduction does not necessarily lower the bit-length of the input and Malikie's proposed construction can lead to incorrect results. For example, Malikie's proposal does not work in the following examples.

- Calculating 5 + 6 mod 19: adding 5 and 6 generates an unreduced result 11. Performing modular reduction on the unreduced result 11 (i.e., 11 mod 19) yields a reduced result of 11, whose bit length is the same as the unreduced result, not lowered.
- Calculating 5 – 6 mod 19: subtracting 6 from 5 generates an unreduced result –1. Performing modular reduction on the unreduced result –1 (i.e., –1 mod 19) yields a reduced result 18, who's bit-length is more than the unreduced result, not lowered.[3]

C. The '827 and '370 Patents

77. The '827 and '370 Patents relate to the prior art Elliptic Curve Digital Signal Algorithm (ECDSA). In ECDSA, the signer selects a long term private key *d*, and computes a long term public key *Q* using $Q = dG$, where *G* is a generator. Ex. E at 2:10-16, 2:28-41. The signer sends a message *M*, and creates a signature, which is comprised of a pair of integers (*r*, *s*). *Id*. at 2:42-49. The signing process includes "comput[ing] a point R=kG that has coordinates (x, y)." *Id*. at 2:49-63. The verifier can take the message *M*, the public key *Q*, and the signature comprising (*r*, *s*) and verify if the message was created by the signer. *Id*. at 2:64-3:4.

[3] Under a typical method to represent negative integers in binary called two's complement, –1 is represented as 11 and 18 is represented as 010010.

78. The asserted claims of the ’370 Patent relate more specifically to public key recovery using a specific equation. Independent claim 1 is a method claim that recites the following steps: (1.A) receiving through a network an electronic message that includes a signature on the message, and the message omits the public key of the signer; (1.B) receiving a first elliptic curve point R associated with a signature component, the signature component includes a first component r and a second component s; (1.C) recovering the public key of the signer using the equation $Q = r^{-1}(sR - eG)$, where e is hash value computed from the message, and G is a generator of an elliptic curve group that includes both R and Q; and (1.D) verifying the received signature using the recovered public key. Ex. F at claim 1.

79. The last phrase in claim 1 of the ’370 Patent recites “which provides an accelerated verification of the received signature.” *Id.* I have been informed that MARA’s position is that this phrase is not limiting and that Malikie’s position is that this phrase is limiting and should be given its plain and ordinary meaning. It is my opinion that if this phrase is limiting, a POSITA would not be able to determine with reasonable certainly the scope of this phrase. That is to say, if the “accelerated verification” is not a result of performing the recited method steps 1.A – 1.D, and therefore requires something more, it is unclear what more is required in order to achieve “accelerated verification” beyond the recited steps 1.A – 1.D. Moreover, the ’370 Patent does not provide objective boundary for determining whether a verification performed using steps 1.A – 1.D is accelerated or not accelerated.

80. The claimed public key recovery method is described in the ’370 Patent at column 15, line 46 to column 16, line 36. The specification states that “[o]mitting the public key from the certificate can save on bandwidth and storage and the verification process described above yields reduced verification times.” *Id.* at 16:18-21. This portion of the specification does not provide

guidance on what additional steps beyond 1.A – 1.D are required to result in accelerated verification, nor does this indicate how one would objectively test to determine whether a verification is accelerated or not accelerated.

81. Other parts of the specification describe techniques for improving verification speed. *See e.g.*, *id.* at 9:9-15 (pre-computing and storing the single point J), 9:50-53 (providing a pre-computed multiple of the public key Q of the signer), 10:29-67 (combining run-time tables with a revised verification equation). However, claim 1 is not directed to any of these techniques, and therefore, these portions of the specification do not inform a POSITA what additional steps are required with respect to claim 1 to result in accelerated verification.

D. The '961 Patent

1. Random Number Generators

82. I understand that claim 1 of the 961 Patent requires "generating a seed value SV from a random number generator." The paragraphs below provide an overview of random number generators in the context of cryptography, and explain the differences that a POSA would understand between random number generators and pseudorandom number generators.

83. A random number generator ("RNG") in the context of cryptography generally refers to a system or algorithm that produces random numbers. In contrast, a computer program that attempts to approximate random number generation is called a "pseudorandom number generator" ("PRNG"). PRNGs purport to mimic randomness by producing a sequence of seemingly random bits using deterministic algorithms performed on a secret starting value called a "seed." *See* Ex. R at 628 ("The input to the PRBG is called the seed, while the output of the PRBG is called a pseudorandom bit sequence.").[4] The deterministic algorithms used by PRNGs

[4] I do not draw a distinction in this Declaration between bit generators ("BG") and number generators ("NG"). Any bit generator may also be considered a number generator, because one

do not produce numbers that are actually random, because using the same seed value(s) as an input will always result in the same output. *Id.* at 629 ("The output of a PRBG is *not* random…," "[d]eterministic here means that given the same initial seed, the generator will always produce the same output"); Ex. S at 3659 ("On modern computers, pseudorandom numbers are generated by completely deterministic algorithms."). Instead, PRNGs approximate randomness by generating pseudorandom values in distributions that are as close to uniform as possible. *Id.* at 3658 ("we discuss the methods which are most widely used, or most promising, for generating sequences of values which try to *imitate such uniform random variables* for simulation purposes. Those sequences are called pseudorandom and the programs which produce them are called pseudorandom number generators.").

84. To achieve true randomness, RNGs rely on a "naturally occurring source of randomness," not deterministic functions or algorithms. Ex. R at 629. "Physical" or "naturally occurring" sources of randomness may include any form of physical entropy, such as thermal noise, atmospheric noise, radioactive decay, recorded audio or video feeds, or even mouse movements on a computer over time. *Id.* at 629-30. While designing RNGs that harness physical randomness to generate random numbers can be more costly and/or slower, many PRNGs can be unsuitable for cryptography applications because they generate predictable sequences. *Id.* at 498-499 ("Since most true sources of random sequences [] come from physical means, they tend to be either costly or slow in their generation. To overcome these methods have been devised to construct pseudorandom sequences…."), 499 ("A plethora of algorithms has been developed to

may generate numbers from a bit generator by simply converting a generated bit string to an integer. *See* Ex. R at 628 ("A random bit generator can be used to generate [] random numbers.").

generate pseudorandom bit sequences of various types. Many of these are ***completely unsuitable for cryptographic purposes***.").

85. Accordingly, a POSA would have been well aware of the distinction in the art between deterministic mathematical functions (PRNGs) and programs using a physical source of randomness (RNGs) by the 1990s. The distinction between RNGs and PRNGs is important for cryptography because of the difference in predictability of results generated between the two, as explained above. *Id.* at 499 ("A plethora of algorithms has been developed to generate pseudorandom bit sequences of various types. Many of these are completely unsuitable for cryptographic purposes."). With an RNG using a source of physical randomness, learning one number in a sequence generated by the RNG would tell an observer nothing about the next number in the sequence because all numbers are equally unpredictable. But in a PRNG, an attacker could eventually collect enough outputs of the sequence to begin to predict subsequent numbers because the sequence is generated via deterministic functions.

86. The distinction between RNGs and PRNGs is reflected in prior art relevant to cryptography. For example, the prior art Digital Signature Standard (DSS) was published by the National Institute of Standards and Technology in the 1990s. DSS is a standard that sets forth algorithms suitable for use in creating cryptographic digital signatures, including the Digital Signature Algorithm (DSA). The earliest version of DSS, which published in 1994, distinguishes between randomly generated and pseudorandomly generated integers (i.e., numbers). Ex. P at 2068 ("x = a randomly ***or*** pseudorandomly generated integer… k = a randomly ***or*** pseudorandomly generated integer,"), 2075 ("Any implementation of the DSA requires the ability to generate random ***or*** pseudorandom integers…."). Similarly, the Handbook for Applied Cryptography, which published in 1996, separately defines the terms "random bit generator" and "pseudorandom

bit generator," and states that "the output" of a pseudorandom bit generator is "*not* random." Ex. R at 628 (emphasis in original). The Handbook for Applied Cryptography explains that random bit generators are used to generate random numbers by generating a bit sequence and then converting it to an integer. *Id.* at 628. Thus, a POSA would understand the terms "random bit generator" and "random number generator" to be functionally synonymous for purposes of the '961 patent because they both produce random numbers. *Id.*

87. Gathering enough physical entropy to ensure truly random outputs for an RNG may be computationally and temporally taxing compared to generating large numbers of pseudo-random digits using a PRNG. To overcome this, hybrid models have been developed in the art, in which an RNG uses physical entropy to non-deterministically generate a seed for a PRNG. *See id.* at 498-499 ("Since most true sources of random sequences [] come from physical means, they tend to be either costly or slow in their generation. To overcome these methods have been devised to construct pseudorandom sequences in a deterministic manner from a shorter random sequence called a seed"); Ex. S at 3661 ("To introduce some real randomness, one can choose this initial state randomly… A generator with a random seed can be viewed as an extensor of randomness… [i]t stretches a short truly random seed into a long sequence of values that is supposed to appear and behave like a true random sequence."). Depending on the specific security requirements of a given application, this hybrid model, in which a PRNG generates random numbers using a high-entropy seed value, may be referred to as an "RNG" or a "cryptographically secure PRNG," because the seed is chosen using a non-deterministic method.

2. Indefiniteness of Claim 7

88. For the reasons explained below, it is my opinion that claim 7 of the 961 Patent fails to inform a POSA with reasonable certainty concerning how to perform the recited method steps.

89. Claim 1 of the '961 Patent requires the following steps:

> generating a seed value SV from a random number generator;
>
> performing a hash function H( ) on said seed value SV to provide an output H(SV);
>
> determining whether said output H(SV) is less than said order q prior to reducing mod q;
>
> accepting said output H(SV) for use as said key k if the value of said output H(SV) is less than said order q;
>
> <u>**rejecting said output H(SV) as said key if said value is not less than said order q;**</u>
>
> <u>**if said output H(SV) is rejected, repeating said method.**</u>

As highlighted above, a POSA would understand that if output H(SV) is rejected in claim 1, "repeating said method" requires repeating the steps of "generating a seed value…," "performing a hash function H() on said seed value SV to provide an output H(SV)," and "determining whether said output H(SV) is less than said order q prior to reducing mod q" to determine whether output H(SV) should be accepted or rejected for use as a key. The process of repeating those steps is reflected in Figure 2 of the 961 Patent and the corresponding text, which describe repeating the steps starting at generating a seed value using a random number generator after rejecting output H(SV) because it is not less than q:

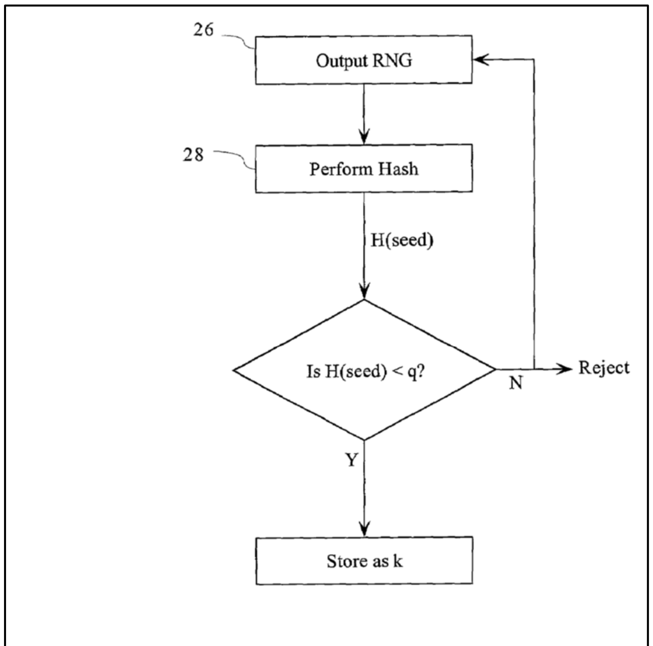


Fig. 2

Ex. G at Figure 2; *see also id.* at 4:13-17 ("If not, the value is rejected and the random number generator is conditioned to generate a new value which is again hashed by the function 28 and tested. This loop continues until a satisfactory value is obtained.").

90. Claim 7 of the 961 Patent, which depends from claim 1 of the 961 Patent, is set forth below:

> 7. The method of claim 1 wherein **if said output is rejected**, said output is incremented by a deterministic function and
>
> a hash function is performed on said incremented output to produce a new output;
>
> a determination being made as to whether said new output is acceptable as a key.

As shown above, claim 7 sets forth a different set of steps to be performed if "said output," which refers to output H(SV), is rejected. Specifically, claim 7 requires incrementing a rejected output by a deterministic function and hashing the incremented output to produce a new output, which is then assessed for its acceptability as a key. The different sets of steps set forth in claims 1 and 7 are summarized in the table below:

Steps after Rejection	**Claim 1**	**Claim 7**
Step 1	Generate new seed value (SV) from an RNG	Increment output H(SV) by a deterministic function
Step 2	Hash function H() is performed on new seed value (SV) to generate a new output H(SV)	Hash function H() is performed on incremented output to produce a new output
Step 3	New output H(SV) is accepted as a key if less than order q and rejected if not less than order q	A determination is made whether the new output is acceptable as a key

91. I have been informed by counsel that because claim 7 depends from claim 1, practicing the method of claim 7 requires practicing all the steps of claim 1 as well as the steps of claim 7. Claim 7, however, does not describe how to perform the two different sets of steps in the same method if output H(SV) is rejected, and therefore fails to provide a POSA with reasonable certainty concerning how to perform the method required by claim 7.

92. I have also reviewed the specification of the ’961 Patent to assess whether it describes how to perform the different sets of steps together in the same method if output H(SV) is rejected. Based on my review, the specification likewise fails to provide a POSA with any reasonable certainty regarding how to perform the different sets of steps together if output H(SV) is rejected. Notably, the specification appears to only describe the different sets of steps as alternatives if output H(SV) is rejected, and not as steps to be performed together as part of the same method. For example, Figure 2 of the ’961 Patent illustrates repeating the steps of the method set forth in claim 1 (including generating a new seed value from an RNG that is subsequently hashed), while Figure 3 illustrates a variation[5] of incrementing a rejected output to produce a new

[5] I refer to claim 7 as a variation of incrementing a rejected output because claim 7 requires incrementing “said output” (*i.e.*, output H(SV)), while the specification describes incrementing a

value that is subsequently hashed as set forth in claim 7. Ex. G at Fig. 2, 3. The text of the specification likewise describes repeating the random number generation step (claim 1) and incrementing a rejected output (claim 7) as alternatives, and not steps to perform together as part of the same method. *Id.* at 4:50-52 ("Upon rejection, the random number generator may generate a new value as disclosed in FIG. 2 ***or*** may increment the seed value as disclosed in FIG. 3.").

93. Given that claim 7 requires performing both sets of steps as part of the same method while the specification only describes them as alternatives, a POSA would not know with reasonable certainty how to perform the steps together as set forth in claim 7. For example, a POSA would not know whether claim 7 requires (1) producing only one new output H(SV) by combining both sets of steps in some undisclosed way; (2) producing only one new output H(SV) by hashing a new seed value SV generated from an RNG; (3) producing only one new output by hashing an incremented rejected output; (4) producing two different outputs to generate two different candidate keys; or (5) some variation of these four alternatives. Accordingly, a POSA would not understand the scope of claim 7 with reasonable certainty.

I declare under penalty of perjury that to the best of my knowledge the foregoing is true and correct.

Date: 12/17/25

[illegible]

Dr. Çetin Kaya Koç

seed value after output H(SV) is rejected. *See* Ex. G at 4:26-27 ("The output may be incremented by adding a particular value to the seed value at each iteration….").